OpenHarmony

开放原子开源基金会 OpenHarmony开发者大会 2023

OpenHarmony 性能关键架构与体系构筑

高曦

OpenHarmony性能测试专家

王俊涛

OpenHarmony测试工具领域专家

OpenHarmony测试sig组成员

目录 Contents

目录 Contents

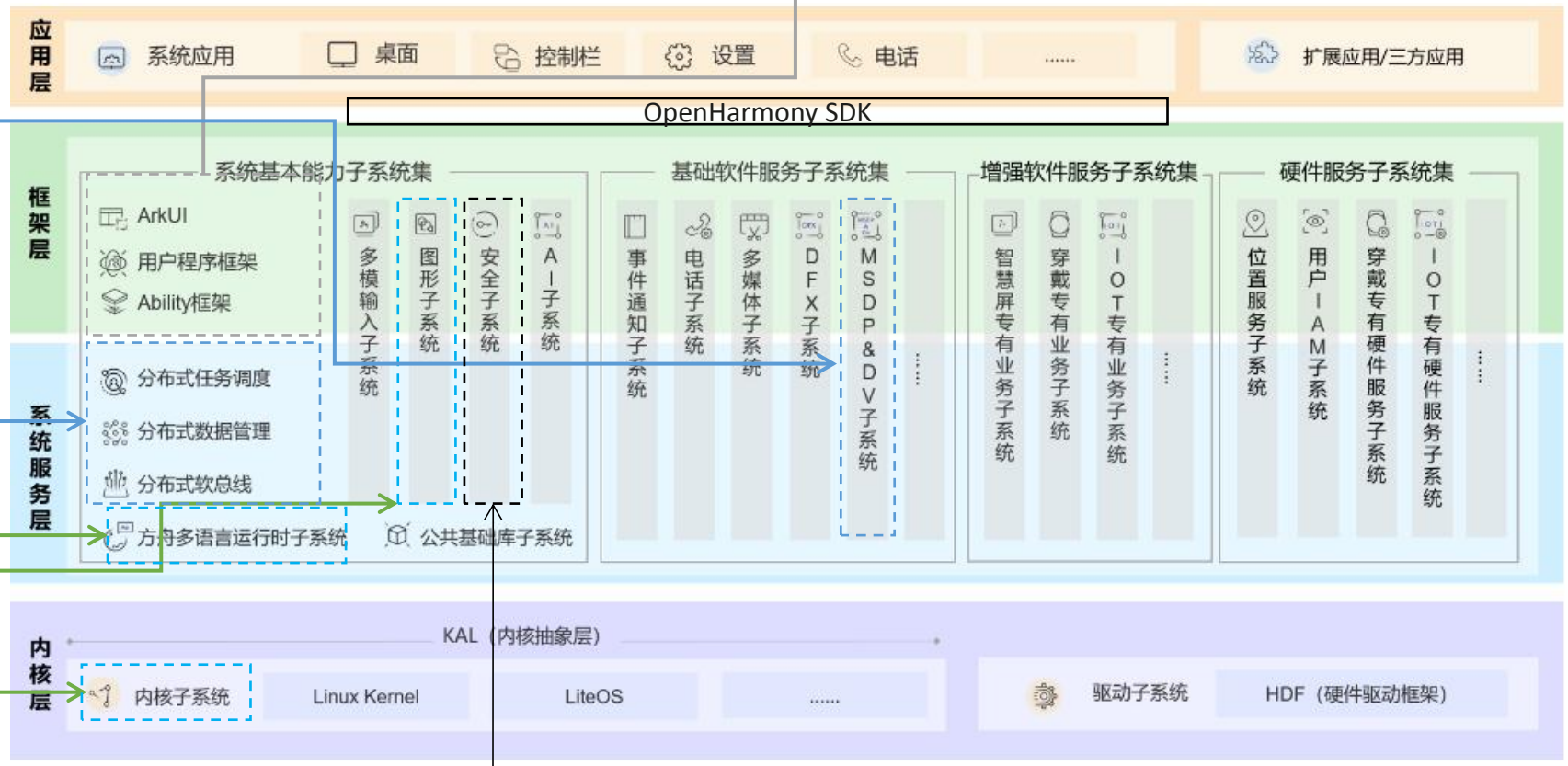
OpenHarmony3.2关键架构演进

OpenHarmony

应用&设备开发演进
一多效率提升
部件化拆解拼装部署
场景化性能分析/调优

分布式演进
无感连接
全栈式流转框架

能效演进
图形高帧率
动效流畅
应用启动减少卡顿
ArkUI最小化更新



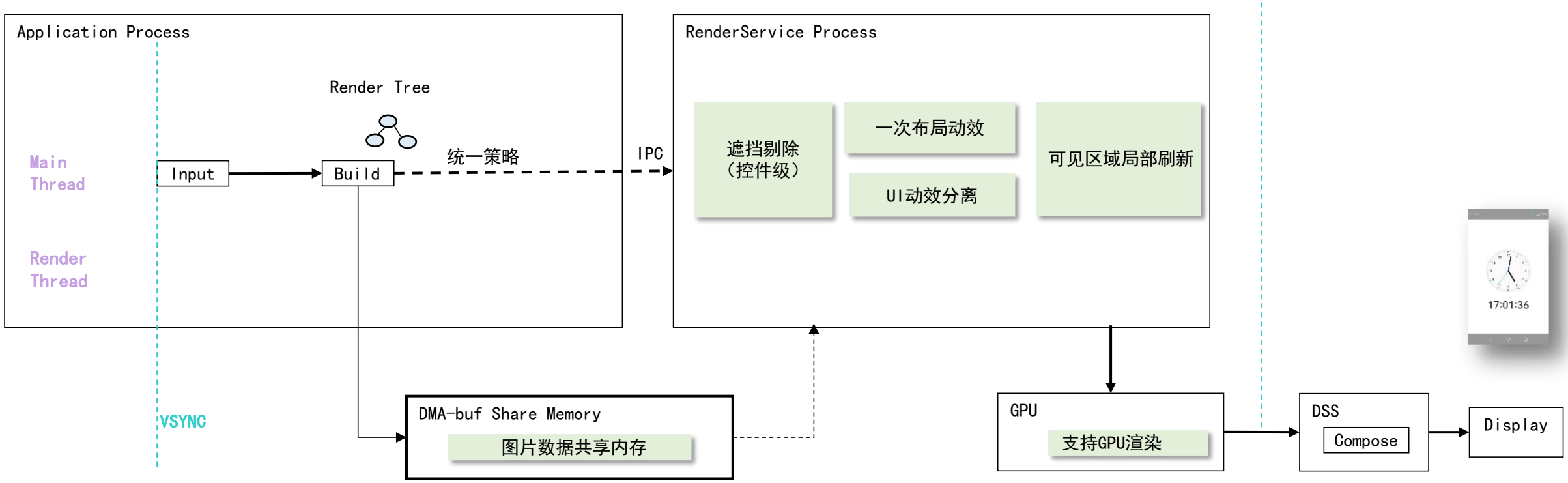
安全演进
DLP文档权限管理

RenderService关键架构介绍

OpenHarmony

架构优势

- 控件级遮挡剔除策略，简化渲染树的结构，结合局部刷新策略 减少渲染负载开销
- AMS/WMS解耦，动效UI分离，一次布局动效策略提升动效性能



控件级遮挡剔除、局部渲染、动效UI分离、共享内存等，提升绘制效率，在部分多窗重载场景下功耗降低10%+

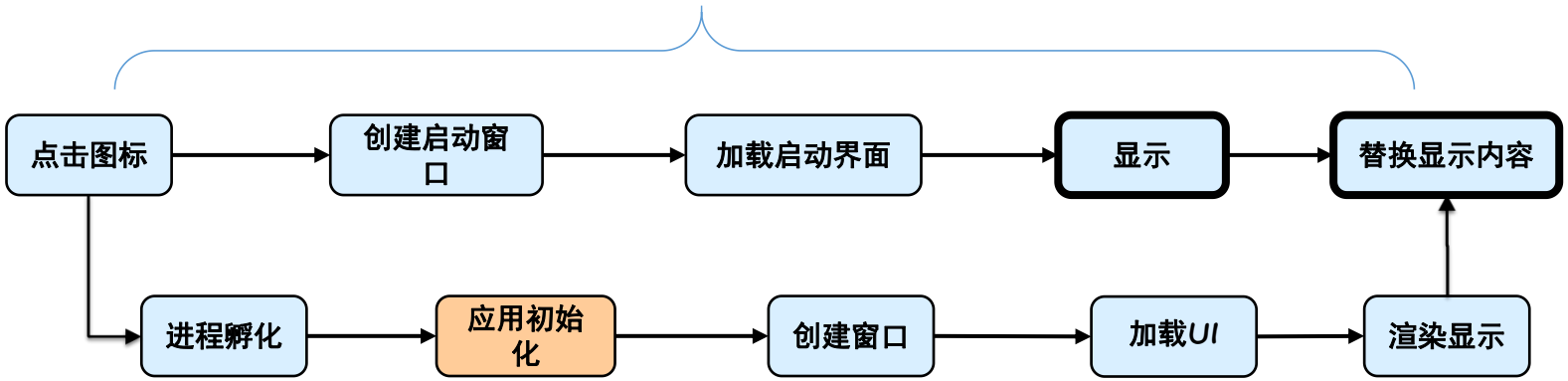
动效流畅关键架构技术点

OpenHarmony

窗口启动对比

快速启动窗口流程：

窗口系统和应用进程并行执行，应用启动可打断始终保持应用极速启动



收益验证：

1. 应用启动退出可打断

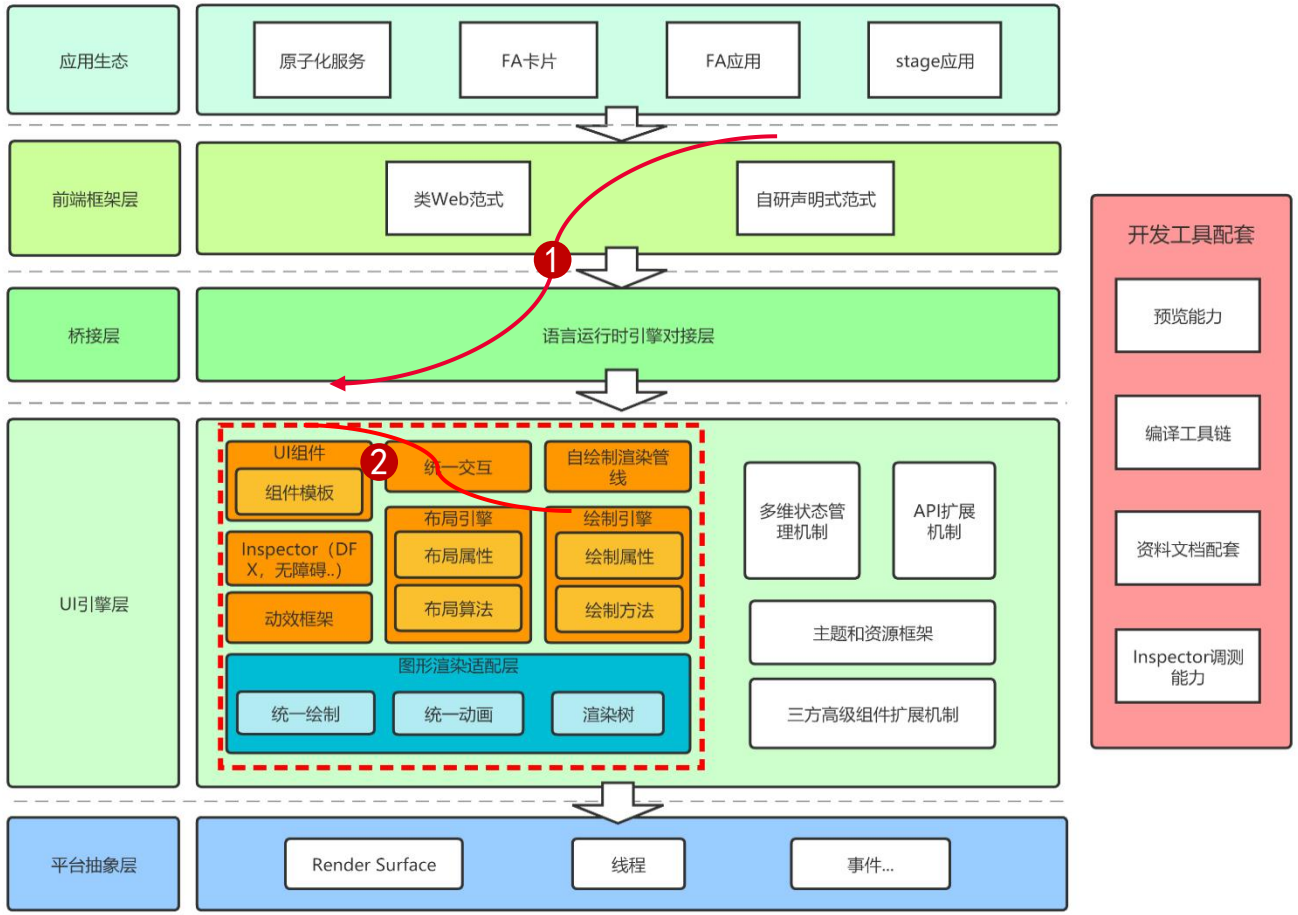
2. 存在初始化回调逻辑的情况下，启动响应时延优化前后收益 提升 20%+

ArkUI最小化更新

OpenHarmony

架构设计

架构目标：组件内存优化80%，应用进程内存较老框架差值不低于组件内存优化；应用其他场景性能数据不劣化



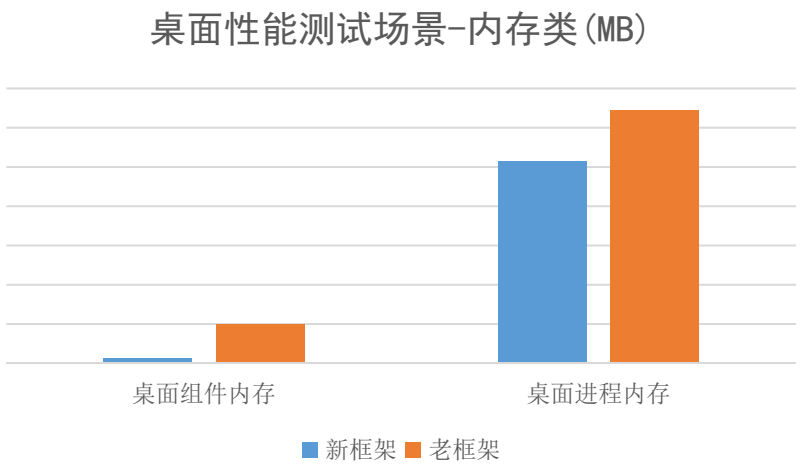
架构优势

ArkUI最小化更新通过UI组件树和节点树优化，达成性能和内存收益

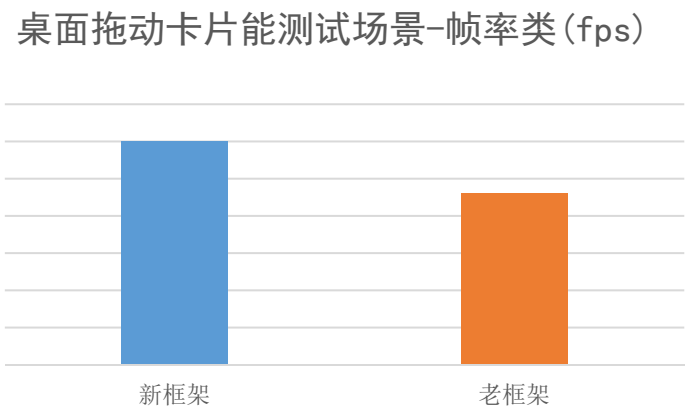
关键架构	关键技术点	优势
最小化更新	前端框架Diff优化	相对比传统声明式UI框架，具有极简的UI树形结构，更短的更新路径，更少的组件内存占用
	三树合并	
	单组件节点结构优化	

ArkUI最小化更新架构收益

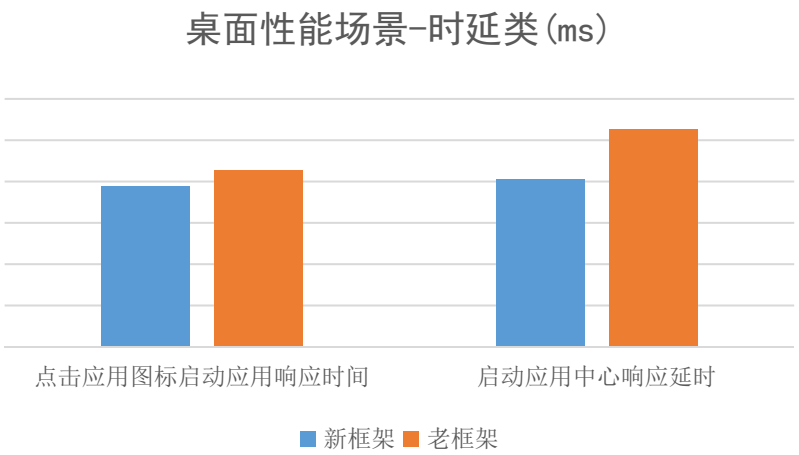
- ArkUI新框架对于应用内存、帧率和时延均有一定程度的提升，其中UI组件树形结构优化对内存有益，最小化更新对时延和帧率有益



桌面内存下降20%



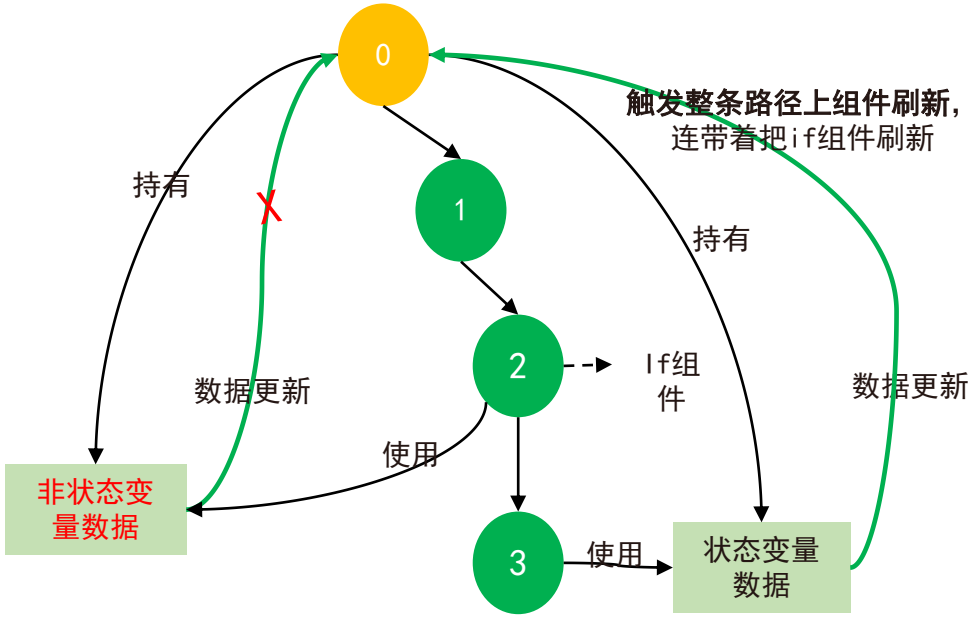
部分场景帧率提升5 + fps

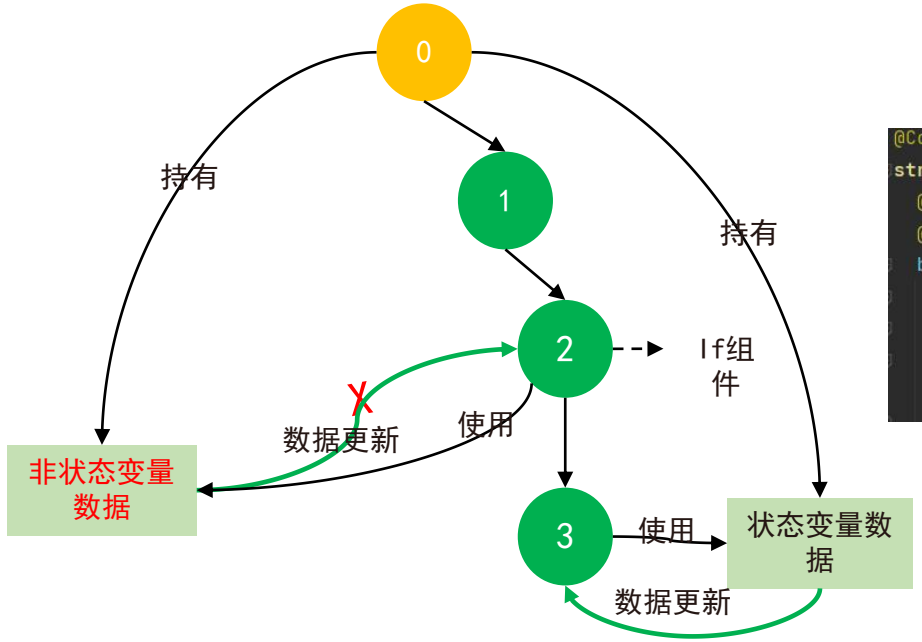


启动响应时延降低8%

ArkUI最小化更新架构带来的变化

应用场景：监听某项数据变化，若数据为true时，则显示刷新组件





定义状态变量：datainfos/finished

```
@Component
struct TestExample {
  @State dataInfos:string[] = []
  @StorageLink('finished') finished : boolean = false
  build() {
    Column() {
      if (this.finished) {
        Swiper() {
          ForEach(this.dataInfos, (item) => {})
```

X 传统架构上未使用状态变量可以刷新组件2和组件3

X 新架构上未使用状态变量导致2无法刷新

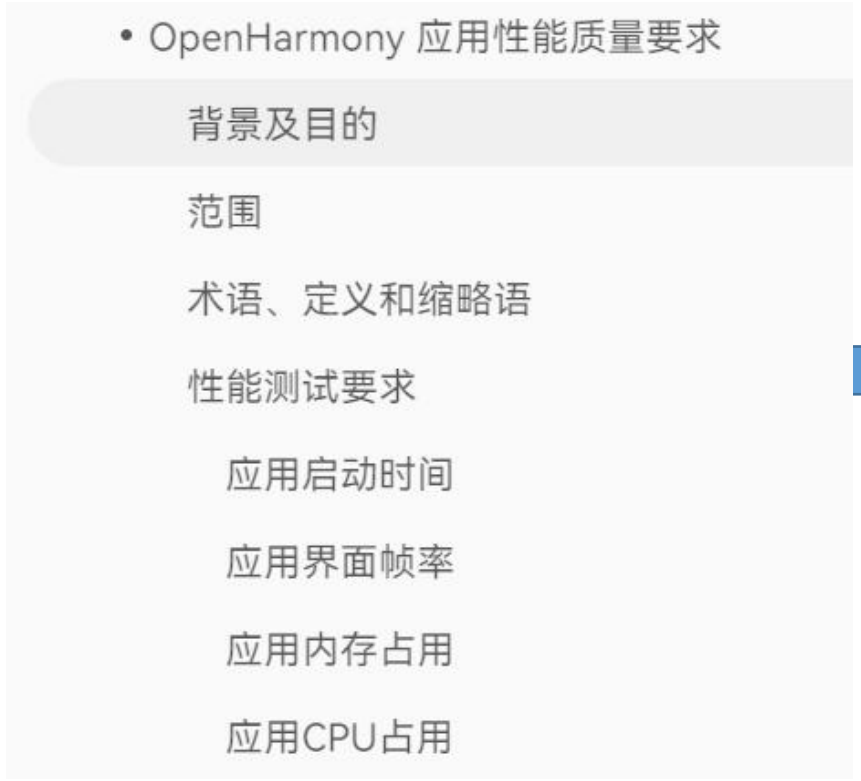
✓ 正确使用状态变量保证组件2、3刷新

- **传统架构以自定义组件为单位进行全量刷新**，只有其他状态变量更新，就能触发整条自定义组件路径上的组件刷新
- **最小化更新架构以组件为单位进行刷新**，需要正确使用状态变量，否则无法达到用户预期效果

目录 Contents

如何评价应用性能

- 针对用户使用场景，定制性能性能标准，包含：应用启动时延、界面滑动帧率、CPU占用、内存占用



类型	测试场景	详细指标
应用启动时间	热启动完成时延	各类应用（不含游戏类、影音娱乐）的热启动时间应≤ 600毫秒 游戏类应用热启动时间≤ 1000毫秒； 影音娱乐类应用热启动时间≤ 1000毫秒； 界面加载过程中白屏/黑屏停留时间≤ 200毫秒。
	冷启动完成时延	各类应用（不含游戏类、影音娱乐）的冷启动时间应≤ 2000毫秒； 游戏类应用冷启动时间≤ 3000毫秒； 影音娱乐类应用冷启动时间≤ 3000毫秒； 界面加载过程中白屏/黑屏停留时间≤ 200毫秒。
应用界面帧率	应用界面帧率	界面滑动或者动态页面刷新帧率（FPS）： 应用的帧率应 ≥ 91.7%*满帧。
应用内存占用	前台内存占用	各类应用在前台1分钟的内存占用应≤ 500MB。
	后台且亮屏内存占用	各类应用在后台且亮屏5分钟的内存占用应≤ 400MB。
	后台且灭屏内存占用	各类应用在后台且灭屏1分钟的内存占用应≤ 400MB
应用CPU占用	后台且亮屏CPU占用率	各类应用在后台且亮屏5分钟的CPU占用应≤ 2%。
	后台且灭屏CPU占用率	各类应用在后台且亮屏1分钟的CPU占用应≤ 2%。

性能质量要求：https://www.openharmony.cn/certification/propertyQuality

如何开发高性能应用

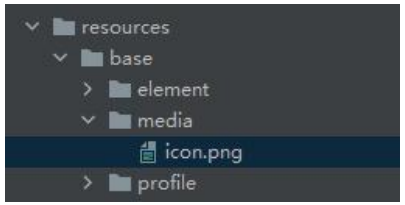
- 为了提高应用性能，提供更加优质用户体验，提供了优化应用时延以及减少丢帧卡顿的一些方法供北向应用开发者参考

	优化类型	优化方向	优化策略	详细案例
性能续航开发实践	时延优化	优化应用启动时间	优化应用进程创建阶段启动时间	1. 设置合适分辨率的startWindowIcon启动图标
			优化Ability初始化阶段启动时间	1. 按需加载依赖，减少不必要的模块加载
			优化Ability生命周期阶段启动时间	1. UIAbility生命周期回调接口里不做耗时操作
			优化加载绘制首页阶段启动时间	1. 异步获取网络数据，不阻塞主线程UI绘制
		提升响应速度	减少UI线程阻塞	1. 创建异步任务处理网络请求 2. 选择合适的组件占位符提升交互体验 3. 使用worker /TaskPool来创建线程并避免线程过多
			减少组件刷新	1. 使用容器限制刷新范围 2. 按需加载列表组件元素
	卡顿优化	减少丢帧卡顿	减少滑动丢帧	1. 减少嵌套层次
			减少动画丢帧	1. 使用系统提供的动效API
	内存优化	减少内存占用	使用系统管理应用内存	1. 后续开放
	后台运行优化	规范任务使用	合理使用后台长时任务	1. 合理使用长时任务、短时任务、延迟调度任务

后续会对社区开发者正式发布，提供真实案例

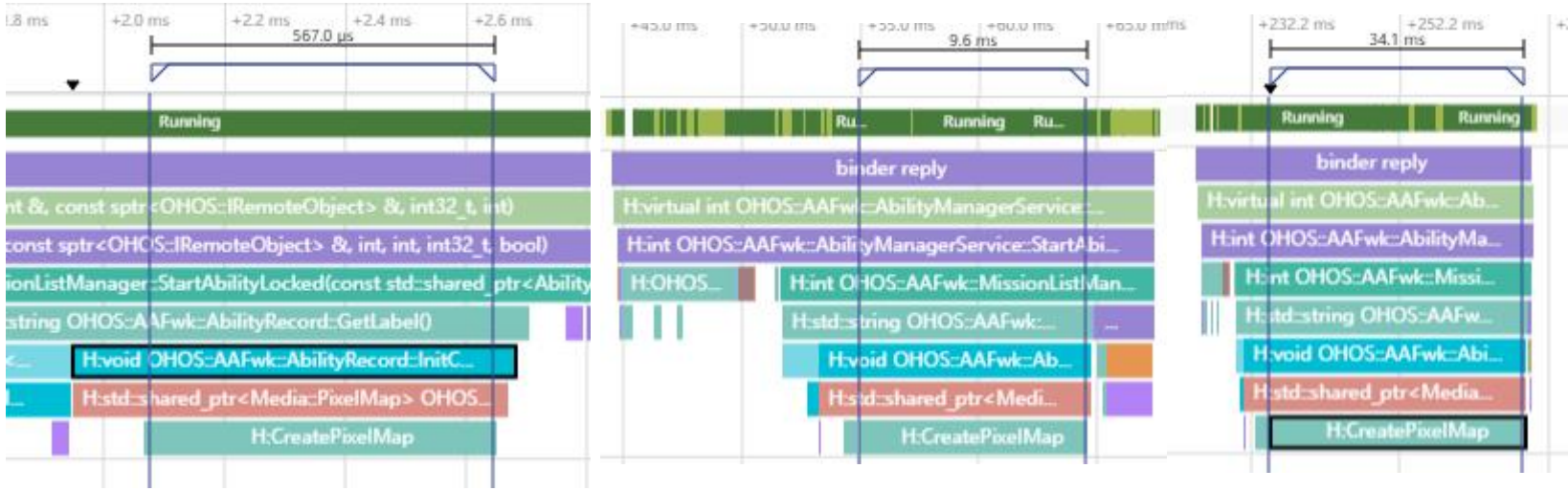
案例：优化应用启动时间——设置合适分辨率的startWindowIcon启动页图标

步骤1：开发者在资源文件夹里放入图标图片



步骤2：在json文件里配置启动页图标

```
"abilities": [
  {
    "name": "EntryAbility",
    "srcEntrance": "./ets/entryability/EntryAbility.ts",
    "description": "$string:EntryAbility_desc",
    "icon": "$media:icon",
    "label": "$string:EntryAbility_label",
    "startWindowIcon": "$media:icon"   // 在这里配置启动图标
    "startWindowBackground": "$color:start_window_background",
    "visible": true,
    "skills": [
      {
        "entities": [
          "entity.system.home"
        ],
        "actions": [
          "action.system.home"
        ]
      }
    ]
  }
]
```



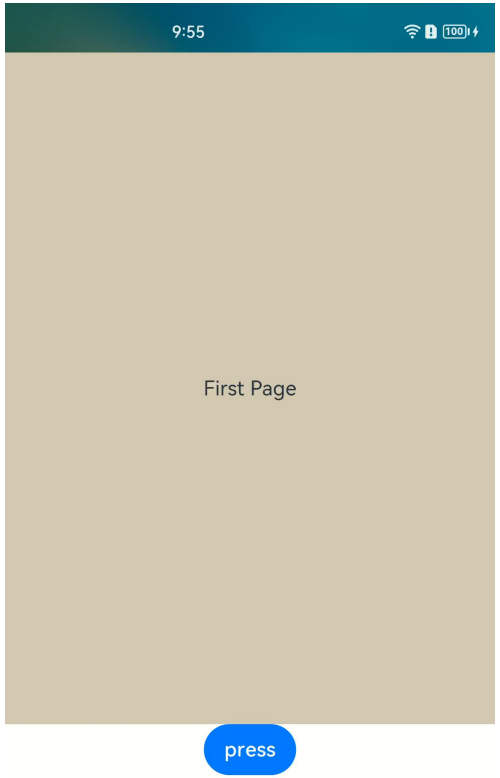
不同分辨率图标对于解码的耗时图统计：

启动图标分辨率	解码耗时（ms）
114*114（模板生成）	< 1
512*512	10
1024*1024	34

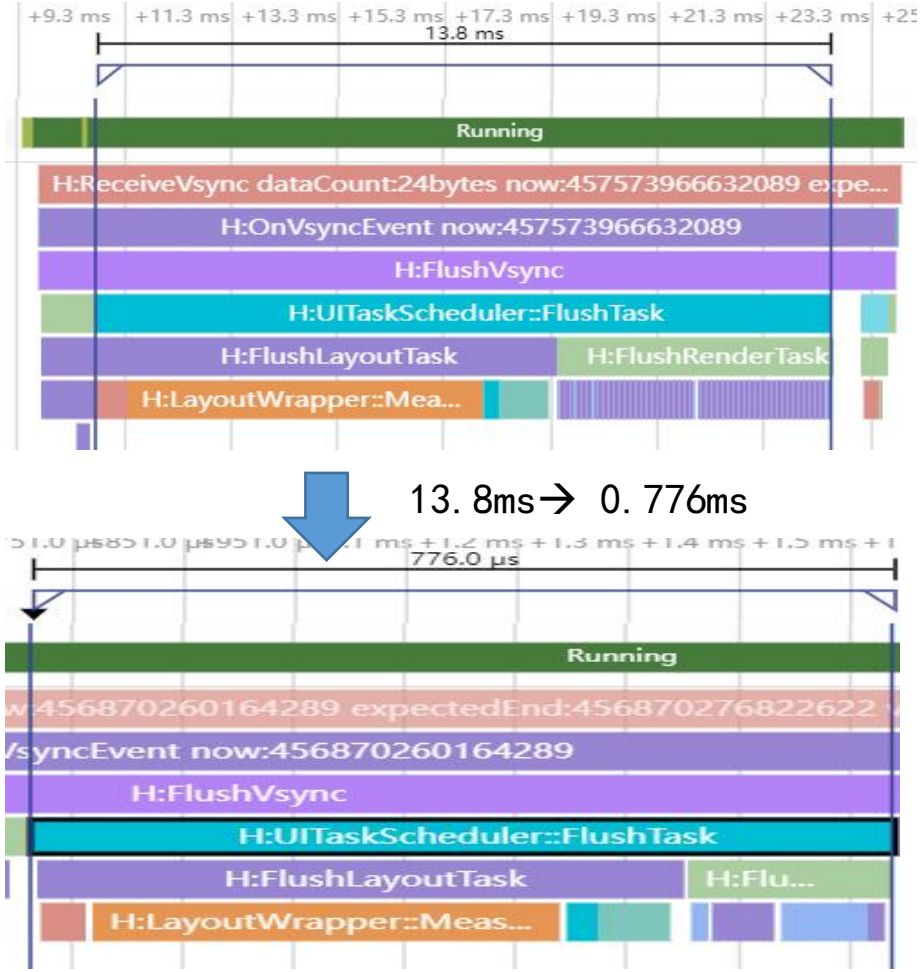
建议开发者设置启动页图标时，分辨率不要超过256*256

案例：提升响应速度——通过stack组件隔离减少组件刷新个数与时间

场景：显示‘Second Page’的Text组件，在状态变量isVisible为false时不显示，而在true时显示。
问题描述：创建和销毁该Text组件时，外层stack容器内的所有组件都会刷新



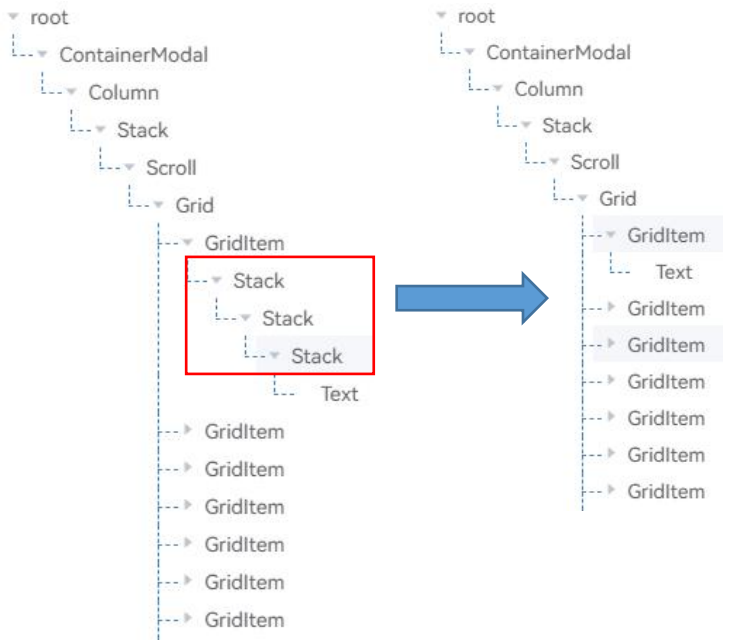

```
@Entry
@Component
struct StackExample {
  @State isVisible : boolean = false;

  build() {
    Column() {
      Stack({alignContent: Alignment.Top}) {
        Text('background').width('100%').height('70%').backgroundColor(0xd2cab3)
          .align(Alignment.Center).textAlign(TextAlign.Center);

        // 此处省略100个相同的背景Text组件

        Stack() {
          Text('First Page').width('100%').height('70%').backgroundColor(0xd2cab3)
            .align(Alignment.Center).textAlign(TextAlign.Center);
          if (this.isVisible) {
            Text('Second Page').height("100%").height("70%").backgroundColor(0xd2cab3)
              .align(Alignment.Center).textAlign(TextAlign.Center);
          }
        }.width('100%').height('70%')
      }
      Button("press").onClick(() => {
        this.isVisible = !(this.isVisible);
      })
    }
  }
}
```



优化方案：定义Stack组件进行隔离，避免外层stack组件内其他组件的全量刷新

案例：减少丢帧——减少视图嵌套层次提升绘制效率

场景： 使用Grid来实现一个网格，每个网格中使用Text组件显示一个数字。
问题描述：satck节点外层套了不必要的3层stack容器，这种情况使得每个GridItem内的组件节点数增加了3倍，会增加UI线程布局的时间，以及渲染线程的耗时



优化方案：优化组件布局到最简，减少stack层级嵌套

目录 Contents

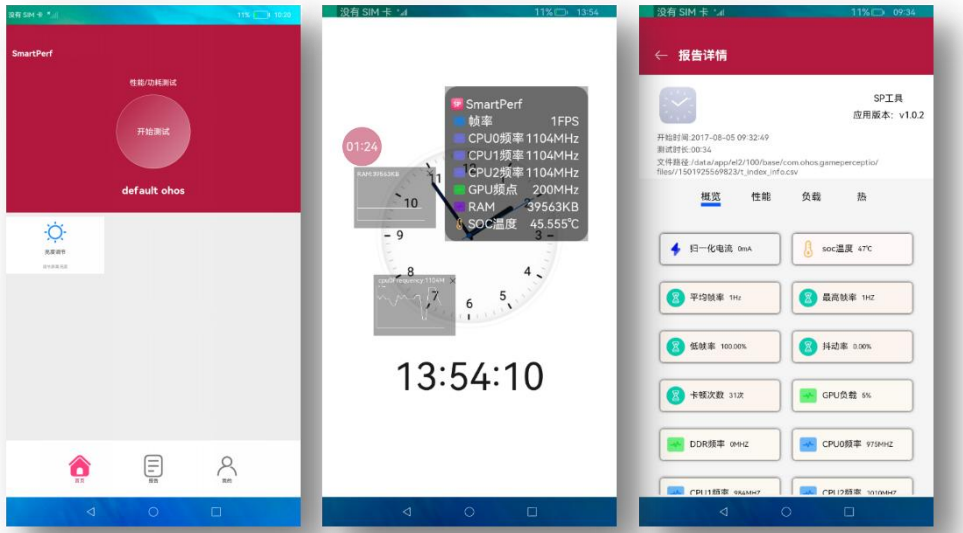
SmartPerf性能测试调优工具

OpenHarmony

在设备端，通过SP-Device，实时采集系统和应用在运行时FPS、CPU、内存数据，结合冷热启动等测试模型抓取关键问题trace，提供关键场景录制回放功能。

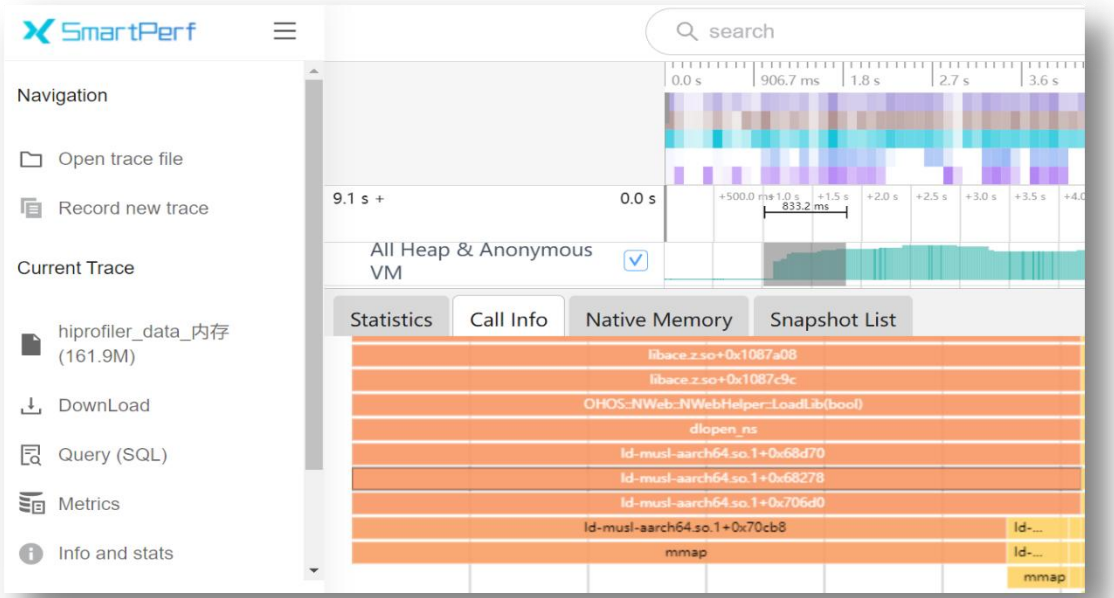
数据采集

SP-Device：性能采集工具



定界定位

SP-Host：性能调优工具



在云端，通过SP-host，为开发人员提供细粒度的问题定位定界手段，像卡顿丢帧，响应时间，内存超基线等问题，通过trace和调用栈进行分析，已经可以拆解到每个CPU上的进程/线程以及so粒度，辅助开发人员定界定位。

SmartPerf-Device 设备侧性能采集工具

OpenHarmony

SmartPerf-Device功能和使用场景介绍

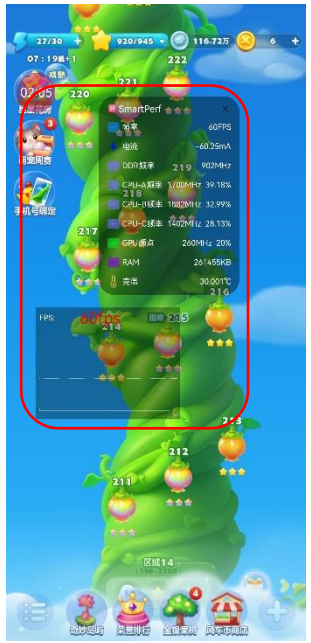
性能数据采集，包括FPS、CPU频率、内存数据、温度、Trace、功耗抓取：

1、分析流畅度，通过实时采集FPS，提供抖动率、丢帧数等指标

2、通过应用运行过程中抓取的trace分析分析应用启动、页面切换时延

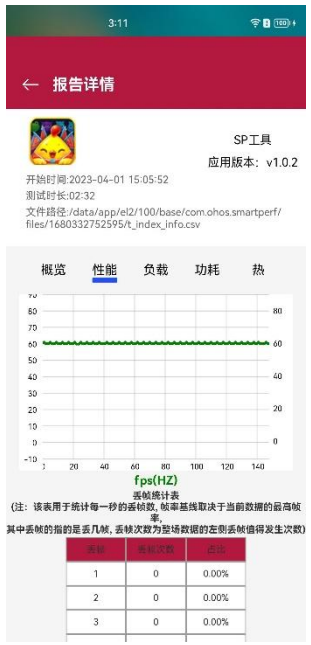
3、通过CPU频点、利用率、温度等信息分析应用和游戏在运行过程中产生高负载、高热的场景

4、测试应用运行中的功耗，定位可能存在问题的进/线程，覆盖整机功耗、Top20进程功耗、应用内线程级功耗

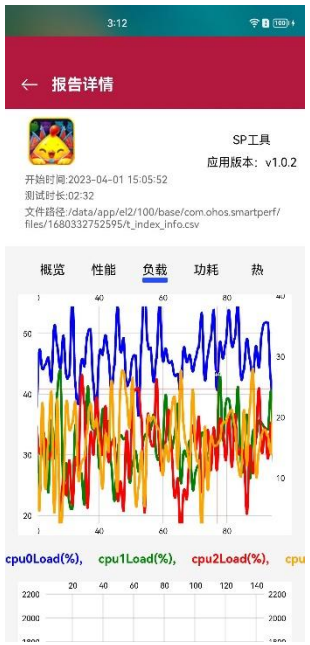


测试实时数据展示

测试报告详情页—概览页



测试报告详情页—性能页



测试报告详情页—负载页

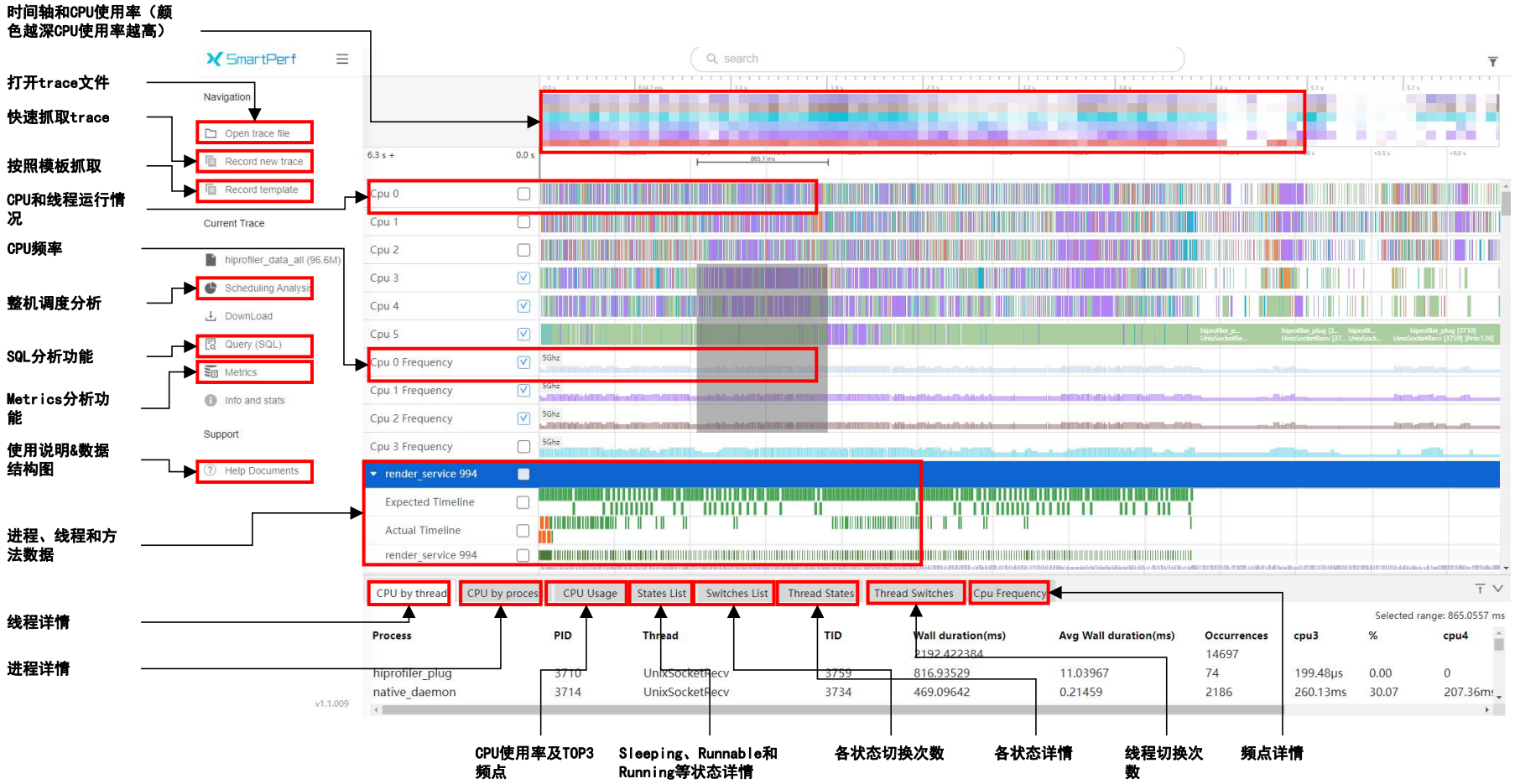
测试报告详情页—功耗页

测试报告详情页—热

Smartperf-Host性能调优工具介绍

工具功能总览

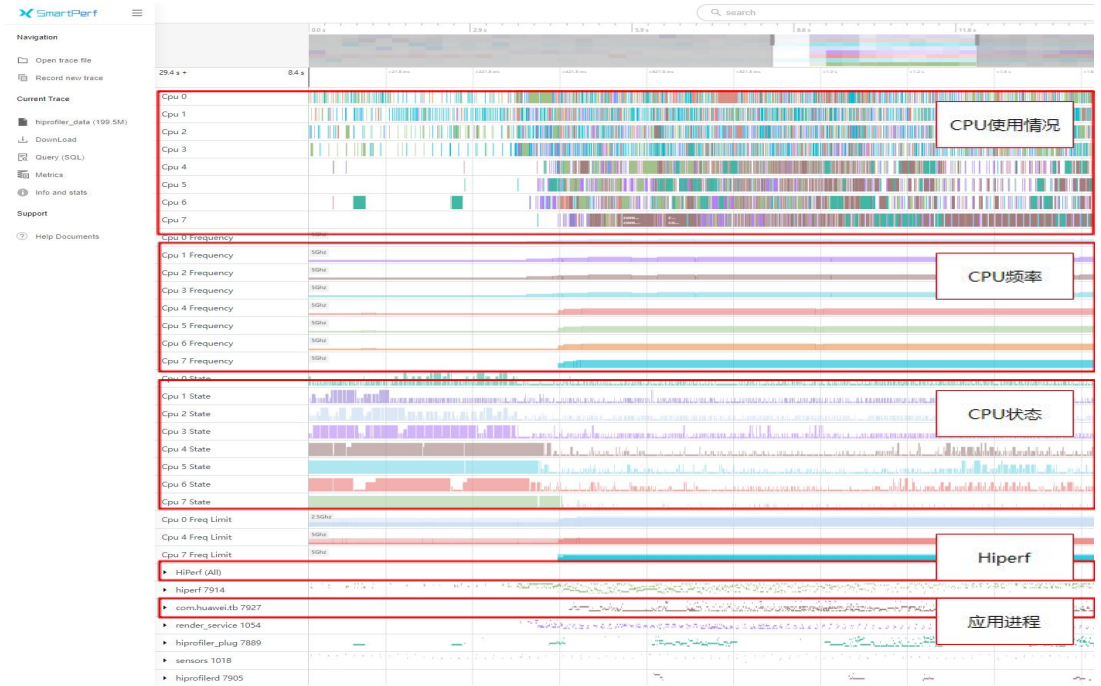
- 页面内Trace抓取
- CPU使用情况展示
- NativeMemory内存抓取及分析
- 文件系统跟踪
- 支持HiPerf的采样展示（CPU 、进程调用栈）
- 细粒度性能调优分析 （框选+自定义SQL）



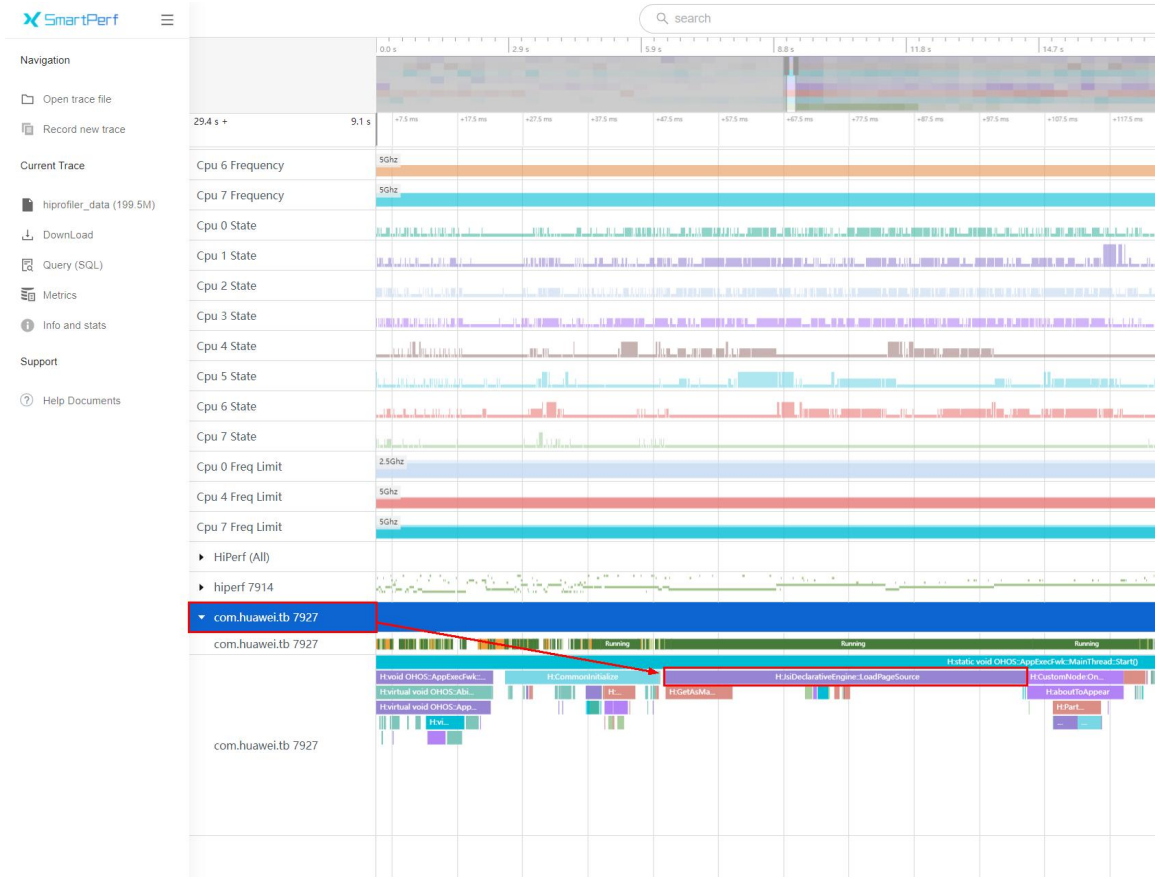
案例：应用启动时间长问题分析

➢ 通过整机调度分析和Hitrace 线程泳道图中发现超长运行线程

1、分析问题出现点的CPU和频点正常：中核，2.0GHz



2. 展开应用进程泳道图，在线程泳道图中可定位运行时间较长的调用栈（但无法看到定位深层的函数，需使用hiperf进一步分析）



注：Cpu使用情况、频率、状态、进程和线程泳道图为hitrace数据，Hiperf(All)为hiperf数据

案例：应用启动时间长问题分析

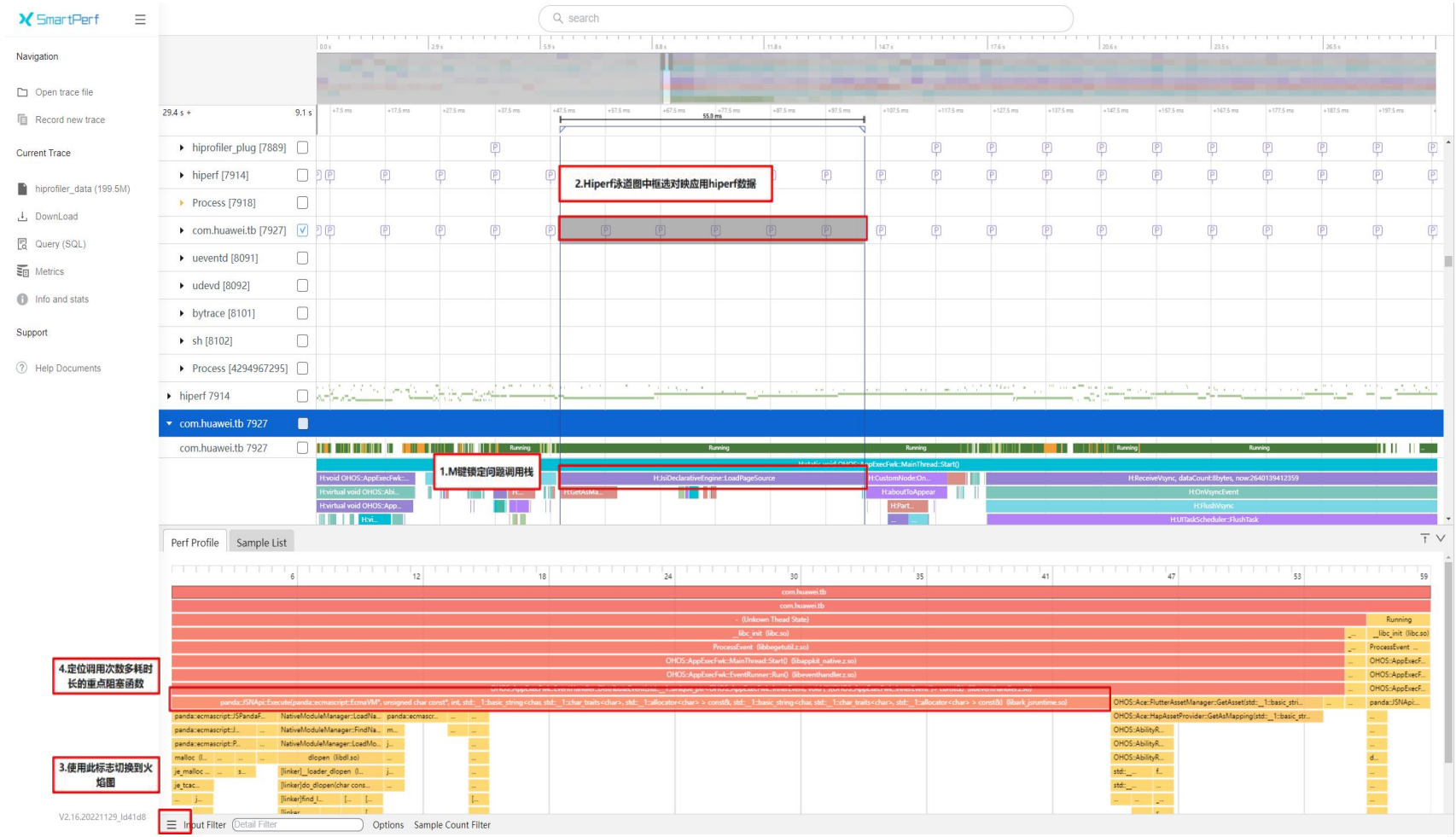
通过Hiperf数据发现一个jsNapi执行时间太长，转而对该api的逻辑进行分析

- Hiperf 中定位阻塞函数

1. M键锁定问题调用栈
2. Hiperf泳道图框选住对应位置
3. 使用火焰图定位阻塞函数

火焰图介绍：

1. 每一列代表一个调用栈，每个格子代表一个函数
2. 纵轴表示调用栈的深度，按照调用关系从上到下排列
3. 格子的宽度代表其在采样中出现的频率，宽度越大，说明是阻塞原因的可能性越大



OpenHarmony

THANK YOU

扫描二维码 关注官方公众号

【官网网址】www.openharmony.cn